MC/DC - TNT:

Monte Carlo Dynamic Code - Toy Neutronics Testbed

An event based transient Monte Carlo testbed for investigations into metaprograming techniques in Python, based on methods from MC/DC

Jackson P. Morgan

Center for Exascale Monte-Carlo Transport (CEMeNT), Oregon State University

December 15th 2021

Contents

1 Introduction

1.1 Purpose

As part of CEMeNT's objectives of investigating metaprograming techniques for parallelism on heterogeneous exascale compute systems [6], a light implementation of some the methods contained within the MC/DC package has been created. The purpose of this testbed is to to implement various potential pythonic based metaprograming pathways that have been proposed, and ultimately decide which one, or what combination, to implement on MC/DC itself. MC/DC is relatively feature rich and written in a structure that does not lend itself conveniently to vectorized parallelization. Whereas TNT has limited simulation functionality and is written with GPU parallelization in mind.

1.2 Targeted Features

Event based transport

Event based Monte-Carlo methods are commonly implemented to decrease the divergence of a code for better optimized performance on a GPU. This is done by taking all particles through a given event at the same time (i.e. lots of for-loops over the total number of particles alive). This is in contrast to history based transport, the other commonly implemented method, which takes individual particles through their whole *life history* before moving onto a new particle. As Event-based methods take all particles undergoing a given event through the same series of operations with similar data the produced functions are vary readily parallelizable in general and especially for vector based devices.

Slab geometry

Implementing new methods and simulation packages in so called "slab geometry" (aka plane wall geometry) is standard in non-production codes, as it allows method and computational exploration without requiring implementation of complex geometry functions. Particles will still undergo three dimensional transport within the wall but the changes in y and z are unimportant as it is assumed that the materials are homogeneous in that coordinate plane.

Multi-region

Having multiple regions in a neutron transport code means having non-homogeneous material and cross-section data through out the simulation. In TNT this is handled by implementing a uniform mesh over the problem domain, where the mesh cell width is always at least as small as the smallest layer of material. This mesh is also used to calculate things like scalar flux (a reaction rate density), and current throughout the problem domain.

Surface tracking

This is a method of handling multiple material regions where the nearest region boundary is kept track of, and compared to, the distance to collision. If the boundary is closer the particles is moved to the boundary (usually plus a small kicker to avoid computation issues) and a new distance to collision is found. Normally this is done only at interfaces of differing material, in TNT this is done between each

mesh boundary to avoid divergence. We expect the extra overhead of more re-computation of distances to be much smaller than the alternative.

Live fission particle production

Live fission particle production is something that a lot of Monte-Carlo neutron transport codes do not do, and is inherit to the transient method which MC/DC is built upon. This means that when a fission occurs it's prompt (instantaneously generated) neutrons are produced at the same point in time and space that the fission happened. These daughter particles can then go on to have more interactions through out the problem. Implementing this means that the population of the particles undergoing transport is not constant and is changing as the simulation is cycled through. This presents data storage and handling issues that have been handled in TNT 2.

Mono energetic

A simplifying assumption that all neutrons are at the same energy and traveling at the same speed. Generally this is not a good assumption to make as particles produced via fissioning events are at much higher energies then those required to produce fissions in the first place. One of the goals of a reactor is to slow these particles, using moderators, down from the fast energy level to the thermal one where fissions can occur. In most neutron transport codes this is better handled using the multi-group assumption, where material data is produced for discretized energy levels and assumed homogeneous therein. Also continuous energy can be used here. For now TNT make's the same speed assumption to simplify material data handling and some transport features, but could be implemented later.

2 Phase Space and Tracking Parameter Allocation

The phase space is the list of independent parameters that are required to completely nail down a neutrons state. In TNT they are a location (x, y, z), a direction unit vector (i, j, k) and a particle's individual time (t). Also TNT keeps note of other useful tracking parameters, like current mesh cell, speed (the same for all of them), and an alive flag. As TNT is an event based code, it is advantageous to keep these relevant parameters in vectors of single parameters. For example consider a vector of x, and a separate vector of y locations within the problem domain. They are the same length as the number of particles that are currently being tracked, and the index of any of the vectors refers to the individual particle being examined. So x[5] and y[5] are the x and y location of the 5th particle under going an operation.

As particle population itself is now a transient that changes with each event cycle, a method of storing a greater or smaller volume of particles than those initially produced is required. This is done by initializing these particle vectors significantly larger than the initial allocation of particles. Like a water reservoir, the size is allowed to be greater than that of the initial amount so that more particles can flow in and flow out without worrying about an "over topping." To do this a *sufficient volume* of the vectors must chosen.

This sufficient volume can be tricky to find as it is intrinsically related to one thing that TNT will eventually solve for, which is the criticality of the system, k_k. This value is tied to neutron population growth or, lack there of, in a reactor and is defined by

$$k \equiv \frac{\text{rate of neutron production}}{\text{rate of neutron loss}} \tag{1}$$

and is directly related to neutron population by

$$N(t) = N_0 * e^{\frac{k-1}{l}t} \tag{2}$$

where l is the average lifetime of a neutron in the system. When $k > 1$ the system is *super-critical* and the population of the neutrons will grow unbounded (provided material properties etc. are constant), when $k < 1$ the system is *sub-critical* and the population will decay to zero through time, and when k is exactly zero the system is in an equilibrium state or just *critical*. Cirticality can change through time as the system changes. For the purposes of TNT this is all to say that when running problems the user should do one or multiple of the following:

1. Ensure the system is sub or exactly critical
2. Make the maximum simulation time before the time when the reservoirs overflow
3. Make sure the system moves about criticality so the population does not hit the max
4. Implement population control methods (not currently implemented) [1]

For example in problems with no neutron sources, no incident neutrons streaming into the problem domain, and no fissioning events finding this "sufficient size" is easy as it can guaranteed the population is both sub-critical and strictly decreasing. However even when the problem is sub-critical the user needs to ensure that the phase space allocations are larger than the initial allotment for momentary population increases during an event cycle due to how particles are handled after they end there transport journey (section 7). To keep track of the maximum index of the phase space vectors a variable which represents all alive particles is kept.

To enable the greatest compatibility with the various methods that will be implemented on this test-bed each phase space vector and particle tracking vector are all kept separate from one another and no classes are used. While not the prettiest way to implement code, especially in a pythonic framework, we expect it to alleviate integration issues down the pike.

3 Particle Sourcing

Before the transport loop can be started an initial population of particles and their locations within the phase-space must be established. To find an initial position (where the only variation is in x) first a mesh-wise fission probability distribution is constructed using fission cross-section data and normalized. Then TNT samples this distribution to find a birth mesh cell. Finally a randomly selected position within the cell is used to provide the particle with an exact location.

When sourcing direction the user can produce either an isotropic source (random in direction) or a uniform one. Uniform is easy to set as the direction [1, -1] in the $\hat{i}$ direction, and all others complementary to it. If the problem is isotropic in nature two random angles for spherical coordinates are found, a zenith

$$\mu = 2\xi - 1$$

and an azimuth

$$\theta = 2\pi\xi$$

where ξ is a pseudo-random number between 0 and 1 (new for each operation) [4]. These angels are then converted to Cartesian unit vectors via

$$c = \sqrt{1 - \mu^2}$$

$$\hat{i} = \mu$$
$$\hat{j} = \cos(\theta) * c$$
$$\hat{k} = \sin(\theta) * c$$

This is done for each particle in the initial allocation and then move to transport.

4 Surface Tracking Through Mesh

As previously discussed TNT implements a surface tracking algorithm where mesh bounds will be used as region interfaces, even if the material is the same on either side. To do this the particle will continue to move through the mesh till it has a collision or leaves the problem domain, at which time that exit is tallied.

First a distance to collision is sampled using the standard cumulative probability function derived from attenuation

$$||D_{\text{collision}}|| = \frac{-ln(\xi)}{\Sigma_{\text{total, i}}} \tag{3}$$

where $\Sigma_{\text{Total, i}}$ is the total macroscopic cross-section in the i^{th} mesh cell [3]. Next the distance to the nearest mesh bound is found. This can be done by first examining the direction, then calculating the distance to the next bound in that direction. TNT's slab geometry makes this relatively simple. For example if a particle is going to the left then it is known that the left bound of the particles mesh cell is the closest interface, and the particles distance to that bound is given by

$$||D_{\text{bound}}|| = \frac{i * dx - x}{\mu} + \text{kicker} \tag{4}$$

where x is the current location of the particle, i is the known mesh cell, dx is the uniform mesh cell width, and the kicker is a very small distance to move the particle off the cell wall (implemented as 10^{-10}). Note that this distance should be a magnitude, thus always positive.

If it moved cells then the cell value is changed, the particle is moved to its new x, y, z, location, the distance and the distance squared that particle moved is tallied and stored in a mesh-wise accumulator vector (see section 8), and the time is incremented based off the speed and distance gone. This process is then repeated.

If the collision event is closer than the particle is again moved and all the necessary items updated as in the previous case, only now the particle is done with transport and can move onto sampling what collision it had.

5 Sampling Events

In order to sample events a normalized probability distribution function in each cell is built and a random number is generated to determine which event happened. To make this decision the known cross section data for the three events TNT will simulate, is used. A capture event, where the particle is killed without anything else happening, a fission event where that individual particle is killed and a multiple of new particles are generated, and a scattering event where the particle gets isotropicly scattered.

To generate the PDF for a single particle the cross-section data is normalized so that

$$\Sigma_{\text{scatter, norm}} = \Sigma_{\text{scatter}}/\Sigma_{\text{total}}$$

$$\Sigma_{\text{capture, norm}} = \Sigma_{\text{capture}}/\Sigma_{\text{total}}$$
$$\Sigma_{\text{fission, norm}} = \Sigma_{\text{fission}}/\Sigma_{\text{total}}$$

Region dimensions are then produced where the size is weighted to that events cross-section when compared to the others. For example consider the situation where $\Sigma_{\text{scatter, norm}} = 1/4$, $\Sigma_{\text{capture, norm}} = 1/2$, $\Sigma_{\text{fission, norm}} = 1/4$, this step distribution can graphically represent in figure 1. Where a random number rolled between 0 and 1 will define a location in that distribution and thus correspond to the event that happened. Note that the order of these operations in the PDF does not matter, only there normalized sizes will have an impact on the rate of events that are selected.

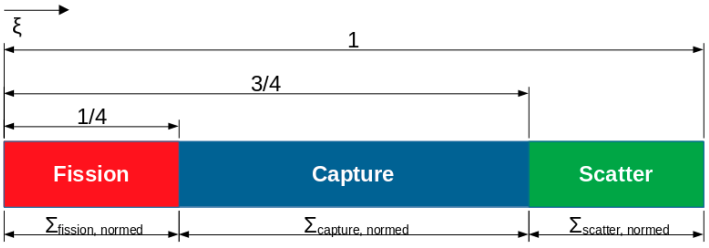

Figure 1: shows how we can select an event using a constructed PDF created by normalized cross-sections and a random number between 0 and 1. In this case say 0.66 is rolled, that would mean a fission event has been selected.

After an event is selected the index of the particle is saved in a vector of indices to be passed to the specific event functions. Also a count is kept for the total number of events in that vector to keep track of the maximum index of this vector of indices.

6 Collision Events

In TNT three collision events have been modeled: scatter, capture, or fission. While being sampled the index of the particle in the phase space vectors, that will undergo that collision operation is stored. When a specific collision function is called it will enter a for-loop over those stored indices. Other ways where initially implemented—like using a single event vector with various integer values corresponding to specific events—but where not ultimately used as they would have increased divergence and hampered parallel implementations.

6.1 Capture

In TNT a capture requires so few operations it is actually handled in the sampling event function. The alive flag for that particle is set to false and it will get removed in the clean up function (section 7).

6.2 Scatter

Scattering is also a relatively simple function. It moves over all particles that had a scatter event (again, this has been recorded as a vector of indices) and a new azimuth and zenith angle are randomly selected, converted to a unit vector, and recorded in the $\hat{i}$, $\hat{j}$, and $\hat{k}$ vectors. This process is exactly like the isotropic sampling seen in section 3.

6.3 Fission

When adding fissions the fact that the phase space vectors have been made larger than the initial allotment of particles comes in handy to store the new fission particles in the free spaces at the back of the vector. This essentially means that new particles are sourced, however now there is no need to sample location as the location of the daughters is the same as where the parent neutron had its preceeding reaction. The same is done with the daughter's current time. Otherwise new directions are isotropically sampled, the alive flag is set to true and then move on. As the parent neutron was used in the reaction we set its alive flag to false.

7 Moping Up: *BringOutYourDead*

In order to control and conserve memory, a function is used at the end of transport that examines the alive flags of all the particles that where alive at the start of the cycle plus those produced via to fission events. An index is started before the for-loop is entered which corresponds to the number of kept particles. At the end of this process this index will become the number of particles alive, and thus the maximum index for particles in the phase space vectors at the start of the next cycle. If a particle is dead nothing happens. If a particle is alive than all of that particle's data is stored at the kept index, essentially moving it forward in the phase space vectors and writing over the dead particle's data.

8 Scalar Flux Estimation with Error

TNT implements a track length estimator where all particles are of equal weight. If the woodcock tracking delta had been implemented a collision estimator would have used instead of surface tracking. First while the particles are undergoing transport the total distance all the individual particles moved in a given cell is accumulate (D_i). This tally is done in the surface tracking function as that is where the distance traveled in a given mesh cell is readily available. There the the square of the distance traveled in each cell is also accumulate (D_i^2) which will be used to provide a variance of the solution. When transport is complete the average scalar flux in a cell ($\bar{\phi}_i$) can be found

$$\bar{\phi}_i = \frac{D_i}{Ndx} \tag{5}$$

where D_i is the total distance accumulated in the i^{th} cell, N is the total number of initially supplied particles, and dx is the cell width [5]. As the total track length accumulation divided by the total number of initially supplied particles is a mean the aforementioned accumulation of the squares is used in TNT to solve for a variance via a known relation

$$\sigma^2 = \frac{\overline{D_i^2} - (\overline{D_i})^2}{N-1} \tag{6}$$

where $\overline{D_i^2} = D_i^2/N$, $\overline{D_i} = D_i/N$, and N is the total number of initial particles [7]. The difference in the terms is subtle but the distinction is very important for this computation. Now the results can be reported without making Todd angry. These values should be small and will decrease as the number of simulated particles increases, all other things being equal. Also this error will increase with a smaller mesh-cell width (dx), all other things being equal.

9 Test Cases and Results

Two test problems where selected for initial confirmation of TNT. They where compared to solutions provided from MCDC to gain confidence that the event based algorithm was working. Both followed the restrictions of the program—slab geometry, mono-energetic, and where designed to test various features. Schematics of the simple problem geometry can be found in figure2

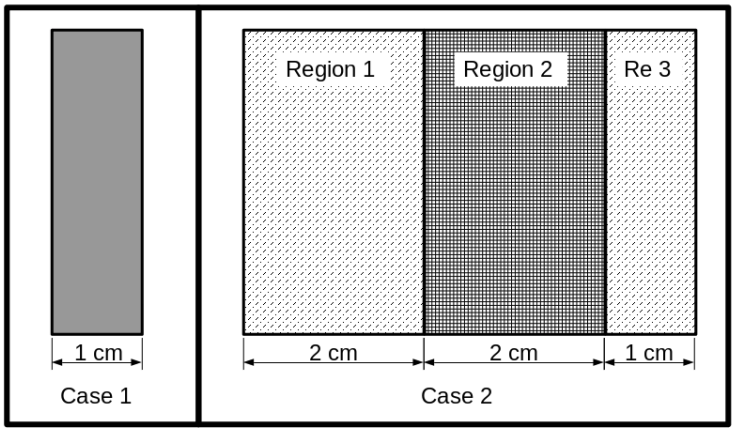

Figure 2: test problem schematics

figures to come

9.1 Case 1

Case one tests the simplest problem that could be conceived with any media. A homogeneous generating region with scattering, capture, and fission cross-sections all being the same ($\Sigma_{capture} = \Sigma_{fission} = \Sigma_{scatter} = 1/3$), and vacuum bounds at the edges. This problem also has the added benefit of having a simple analytical solution readily available for the flux profile via the diffusion approximation to the transport equation and then the geometric buckling method

$$\begin{aligned} \phi_{\text{normed}} &= \cos\frac{\pi x}{\tilde{L}} \\ \tilde{L} &= L + \frac{0.7109}{(\Sigma_{\text{total}} - 0.87/3)} \end{aligned} \tag{7}$$

where x varies [-L/2, L/2] (this can then be plotted [0,L] to lie on top of the TNT solution) [2].

9.2 Case 2

A more complex case to ensure TNT's ability to handle none homogeneous regions was also produced. The three regions have differing cross-sections (presented in table 1).

	Region 1	Region 2	Region 3
$\Sigma_{capture}$	2/3	1/3	2/3
$\Sigma_{scatter}$	1/3	1/3	1/3
$\Sigma_{fission}$	0	1/3	0

Table 1: Material data for regions in case 2

References

[1] Thomas E. Booth. "a Weight (Charge) Conserving Importance-Weighted Comb for Monte Carlo". In: *Radiation Protection & Shielding Topical Meeting* 836 (1996).

[2] J. J. Duderstadt and L. J. Hamilton. *Nuclear Reactor Analysis*. First. New York, NY: Wiley, 1971. ISBN: 0-471-22363-8.

[3] E. E. Lewis and W. F. Miller. *Computational Methods of Neutron Transport*. New York, NY: Wiley, 1984. ISBN: 0-471-09245-2.

[4] Ryan G McClarren. "Chapter 21 - Introduction to Monte Carlo Methods". In: *Computational Nuclear Engineering and Radiological Science Using Python*. Ed. by Ryan G McClarren. Academic Press, 2018, pp. 381–406. ISBN: 978-0-12-812253-2. DOI: https://doi.org/10.1016/B978-0-12-812253-2.00024-8. URL: https://www.sciencedirect.com/science/article/pii/B9780128122532000248.

[5] Ryan G McClarren. "Chapter 22 - Monte Carlo Variance Reduction and Scalar Flux Estimation". In: *Computational Nuclear Engineering and Radiological Science Using Python*. Ed. by Ryan G McClarren. Academic Press, 2018, pp. 407–423. ISBN: 978-0-12-812253-2. DOI: https://doi.org/10.1016/B978-0-12-812253-2.00025-X. URL: https://www.sciencedirect.com/science/article/pii/B978012812253200025X.

[6] Todd S Palmer. *CEMeNT : Center for Exascale Monte Carlo Neutron Transport A PSAAP-III Focused Investigatory Center (FIC)*. 2020.

[7] Paul K Romano et al. "OpenMC: A state-of-the-art Monte Carlo code for research and development". In: *Annals of Nuclear Energy* 82 (2015), pp. 90–97. ISSN: 0306-4549. DOI: https://doi.org/10.1016/j.anucene.2014.07.048. URL: https://www.sciencedirect.com/science/article/pii/S030645491400379X.